Freeform Search

Database: US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term: L29 and (type near definition)

Display: 50 **Documents in Display Format:** - **Starting with Number** 1

Generate: ○ **Hit List** ◉ **Hit Count** ○ **Side by Side** ○ **Image**

Search Clear Interrupt

Search History

DATE: Thursday, October 14, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L30	L29 and (type near definition)	1	L30
L29	L17 and (persistent near data)	25	L29
L28	L27 and layout	16	L28
L27	L26 and instance$	24	L27
L26	L25 and field$	24	L26
L25	L24 and behavior$	24	L25
L24	L23 and (storage (layout or structure))	47	L24
L23	l19 and structure	47	L23
L22	annotating near type near definition	0	L22
L21	L19 and (annotat$)	4	L21
L20	L19 and (storage near layout)	0	L20
L19	L17 and (type near definition)	52	L19
L18	L17 and (tyep near definition)	0	L18
L17	((define or defining) near type near object)	384	L17
L16	defin$ near type near object	782	L16
L15	L2 and l7	0	L15

L14	L11 and l4	0	L14
L13	L11 and (udt)	0	L13
L12	L11 and hydrat$	2	L12
L11	(storage near layout)	666	L11
L10	L9 and metadata	0	L10
L9	L8 and (storage near layout)	6	L9
L8	(user near defined near type) and format	598	L8
L7	(user near defined near type) anf ormat	5944	L7
L6	L4 and (user near defined near type)	0	L6
L5	L4 and udt	0	L5
L4	(persistence near format)	27	L4
L3	L2 and (persistence near format)	0	L3
L2	detect$ near access near pattern	20	L2
L1	redundant near persistence	0	L1

END OF SEARCH HISTORY

L30: Entry 1 of 1 File: USPT Apr 24, 2001

DOCUMENT-IDENTIFIER: US 6223344 B1
TITLE: Apparatus and method for versioning persistent objects

Brief Summary Text (7):
One way in which the performance of application software programs has been improved while the associated development costs have been reduced is by using object-oriented programming concepts. The goal of using object-oriented programming is to create small, reusable sections of program code known as "objects" that can be quickly and easily combined and re-used to create new programs. This is similar to the idea of using the same set of building blocks again and again to create many different structures. The modular and re-usable aspects of objects will typically speed development of new programs, thereby reducing the costs associated with the development cycle. In addition, by creating and re-using a comprehensive set of well-tested objects, a more stable, uniform, and consistent approach to developing new computer programs can be achieved. Closely connected with objects is the concept of "classes" of objects. A class is a formalized definition of a set of like objects. As such, a class can be thought of as an abstraction of the objects or as a definition of a type of object. Each object that is created in an object-oriented system is an instance of a particular class.

Brief Summary Text (9):
Persistent objects usually contain "persistent" data. Persistent data is any data that must exist beyond the lifetime of the process which created it. Although persistent objects are removed from memory when the process which created them is finished, persistent objects and persistent data may be saved on some secondary storage media and can be reconstructed for use at a later time, if and when the object is needed. Persistent objects can be saved on any of several different types of secondary storage media including hard disks, floppy disks or magnetic tape.

Brief Summary Text (10):
The number of accessible persistent objects is not limited to the number of objects that can be created by one process. As mentioned above, one object that has been created by a first process can call procedures or "methods" on other objects that have been created by other, independent processes. Given that many different processes may create and use persistent objects, the number of persistent objects that are accessible in a given object-oriented environment may grow dramatically over time. The collection of objects containing persistent data maintained by an object-oriented system can easily grow to encompass millions of objects.

Brief Summary Text (14):
The versioning method described above, which is typically used with regular objects that do not contain persistent data (herein called non-persistent objects), cannot be efficiently applied to systems containing persistent objects. It is simply not practical to modify and rebuild each persistent object when the number of persistent objects may reach into the millions. The time and system overhead required to rebuild all of the objects can be overwhelming.

Brief Summary Text (16):

Without a mechanism for versioning objects containing persistent data, the use of persistent objects will be limited in application and organizations that utilize distributed object systems will not fully realize the available benefits of persistence and object-oriented programs in distributed object environments.

Detailed Description Text (9):
Another central concept in object-oriented programming is the "class." A class is a template or prototype that defines a type of object. A class outlines the makeup of objects that belong to that class. By defining a class, objects can be created that belong to the class without having to rewrite the entire definition for each new object as it is created. This feature of object-oriented programming promotes the reusability of existing definitions and promotes efficient use of program code.

Detailed Description Text (26):
Referring now to FIGS. 3 and 4, object diagrams will be used to further illustrate and demonstrate the applicability of the present invention. It should be assumed in the following examples that the object depicted is a persistent object such as persistent object 125 as shown in FIG. 1. It should also be assumed that existing related objects reference the persistent object in order to utilize the persistent data belonging to the object. The existing related objects have established pointers which reference the object identity of persistent object shown in FIGS. 3 and 4.

Detailed Description Text (30):
As previously mentioned, the present invention breaks the persistent object into sections. The header section has already been described, and the data section of the persistent object contains the state data belonging to the persistent object. Typical objects that do not contain persistent data, referred to in this specification as non-persistent objects, are not broken up into two sections. Rather, the complete contents (including the object identity and the data) of a non-persistent object are always stored in contiguous locations in memory. Therefore, existing related objects can reference the data section of a non-persistent object using the object identity assigned to the non-persistent object.

Detailed Description Text (43):
Referring now to FIG. 6, a method 600 in accordance with a preferred embodiment of the present invention for creating new state data is shown. As shown in FIG. 6, the old state data contained within the data section of the persistent object may be modified to create the new state data. As previously mentioned, when modifying the state data of the persistent object, the old state data are used to build the new state data. The first step is to determine whether the new state data are derived from the old state data (step 610). If the new state data is not derived from the old state data (step 610=NO), the constructor object is simply called to initialize the new state data (step 615). A constructor object is an object which is capable of creating a new object according to a given class definition. If, however, the new state data are derived from the old state data (step 610=YES), the old state data must be retrieved from storage (step 620). This state data was originally placed in storage in FIG. 5, step 515. The old state data is read (step 625) and the old state data is converted into the new state data (step 630). Those skilled in the art will recognize that there are many techniques which may be employed to accomplish the conversion of the old state data into the new state data. For example, the old object may be "flattened" by streaming the state data out to a byte array in memory. Then after the new object has been created, the byte array may be streamed back into the state data of the new object, where the new object is of a different class than the old object. This is just one example and those skilled in the art will recognize that other techniques are available to accomplish the same results.

CLAIMS:

20. A method for versioning a persistent object to a new version, the new version having new state data, the persistent object having old state data, an old data section and a header section, the old data section storing the old state data, the header section having a data pointer referencing the old data section, the method comprising the steps of:

a) determining if the new state data are derived from the old state data;

b) saving old state data if the new state data are derived from the old state data;

c) determining if the old data section is large enough to hold the new state data;

d) performing the following steps if the old data section is not large enough to hold the new state data:

ii) deallocating the old data section;

ii) allocating a new data section that is large enough to hold the new state data; and

iii) changing the data pointer to reference the new data section;

e) creating new state data; and

f) updating a method table pointer contained in the header section of the persistent object from a first method table to a second method table, the second method table defining at least one new method of the persistent object.

Previous Doc Next Doc Go to Doc#

L21: Entry 2 of 4 File: USPT Mar 6, 2001

DOCUMENT-IDENTIFIER: US 6199100 B1
TITLE: Interactive computer network and method of operation

Brief Summary Text (21):
Additionally, the method preferably includes steps for forming at least some applications with objects and dividing the applications into sections, the objects including display data and/or program code for generating display of respective application sections. Further, the method includes steps for preferably forming the application sections with objects arranged as multiple object types, the object types including: an object type for defining program information used in supporting execution of the respective applications at the respective reception systems; an object type for defining formatting of the respective applications at respective reception system display interfaces; an object type for defining application elements that may appear at respective reception system display interfaces; an object type for defining a make-up template for applications presented at the respective reception system display interfaces; an object type for defining window elements that may appear at respective reception system display interfaces; and an object type for presenting advertising at the respective reception system display interfaces.

Detailed Description Text (75):
[<header>(compression descriptor) (presentation data) . . . (program call) . . . (custom cursor) (custom text) . . . (field definition) . . . (field-level program call) . . . (custom cursor type 2) . . . (custom graphic) . . . (field definition type 2) . . . (array definition) . . . (inventory control)];

Detailed Description Text (78):
[<header>(compression description)<partition definition>(page element call) (presentation data) . . . (program call) . . . (custom cursor) . . . (custom text) . . . (custom cursor type 2) . . . (custom graphic) (field definition) . . . (field level program call) . . . (field definition type 2) . . . (array definition) (inventory control)];

Detailed Description Text (84):
[<header>(compression descriptor) (presentation data) . . . (program call) . . . (custom cursor) . . . (custom text) . . . (field definition) . . . (field-level program call) . . . (custom cursor type 2) . . . (custom graphic) . . . (field definition type 2)... (array definition) . . . (inventory control)];

Detailed Description Text (148):
FIELD DEFINITION TYPE 2 SEGMENT

Detailed Description Text (149):
Field definition type 2 segments 517 are provided to enhance run-time flexibility of fields. Field definition type 2 segment structure is as follows:

Detailed Description Text (150):
where structure is defined below. As with the other segments, "st" describes segment type, and "sl" segment length. Further, "Attributes" describe how the user and RS 400 interact at a rudimentary level. Attributes for field definition type 2

segments 517 are contained in four bytes:

Detailed Description Text (683):
RS protocol defines the way the RS supports user application conversation (input and output) and the way RS 400 processes a partitioned application. Partitioned applications are constructed knowing that this protocol will be supported unless modified by the application. The protocol is illustrated FIG. 6. The boxes in FIG. 6 identify processing states that the RS 400 passes through and the arrows indicate the transitions permitted between the various states and are annotated with the reason for the transition.

CLAIMS:

7. The method of claim 6 wherein arranging the objects as multiple object types includes steps for providing an object type for defining program information used in supporting execution of the respective applications at the respective reception systems.

8. The method of claim 7 wherein arranging the objects as multiple object types includes steps for providing an object type for defining formatting of the respective applications at respective reception system display interfaces.

9. The method of claim 8 wherein arranging the objects as multiple object types includes steps for providing an object type for defining application elements that may appear at respective reception system display interfaces.

10. The method of claim 9 wherein arranging the objects as multiple object types includes steps for providing an object type for defining a make-up template for application sections presented at the respective reception system display interfaces.

11. The method of claim 10 wherein arranging the objects as multiple object types includes steps for providing an object type for defining window elements that may appear at respective reception system display interfaces.

Previous Doc Next Doc Go to Doc#